FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Ferramenta para Text Mining em Textos completos

Hugo José Freixo Rodrigues

DISSERTAÇÃO

Mestrado Integrado em Engenharia Informática e Computação

Orientadores:
Rui Camacho (FEUP)
Célia Talma Gonçalves (ISCAP)

28 de Julho de 2016

Ferramenta para Text Mining em Textos completos

Hugo José Freixo Rodrigues

Mestrado Integrado em Engenharia Informática e Computação

28 de Julho de 2016

Resumo

Vivemos num mundo em constante evolução onde cada vez mais pessoas têm acesso sem restrições à *Internet* e podem divulgar as suas ideias. Isto faz com que existam cada vez mais textos espalhados pela *Web*, sejam estes textos apenas *posts* no Facebook ou artigos sobre as mais diversas áreas. Com uma quantidade de textos tão extensa, são necessárias técnicas que permitam aceder e classificar estes textos de forma rápida e eficaz. *Text Mining* (TM) é uma disciplina que ajuda na análise automática de textos e extrair valiosa informação de grandes corpora.

Com o conjunto de ferramentas poderosas disponíveis para TM é possível interpretar e classificar uma enorme quantidade de textos, de forma a obter informação útil. Atualmente os algoritmos de TM tratam os textos sem considerar a sua estrutura. Um texto é geralmente interpretado pelos algoritmos de TM como um *bag of words*, um conjunto de palavras sem relação entre si. Isto faz com que estes algoritmos tenham por vezes desempenhos mais fracos, ou seja, a qualidade da informação obtida nem sempre é elevada.

Existem diferentes abordagens de TM e vários passos de pré-processamento que podem ser aplicados à classificação de textos completos, mas quais as abordagens que poderão trazer melhores resultados? Quais os passos que devem ser aplicados para que o resultado final seja de elevada qualidade?

Existem hoje em dia, disponível na *Web*, coleções de textos completos. Processar textos completos implica a utilização de recursos computacionais mais poderosos. Nesta dissertação pretendemos avaliar se a utilização de um número reduzido de secções dos textos permite construir classificadores com desempenho semelhante aos classificadores construídos com os textos completos.

Nesta dissertação concluiu-se que os algoritmos de TM, quando aplicados a pequenas secções de documentos, conseguem resultados semelhantes aos obtidos aplicando os mesmo algoritmos a documentos completos.

Abstract

We live in a world in constant change where more and more people have unlimited access to the internet, where they can post their ideas. This means that there are more and more texts available on the Web. These texts can be simple Facebook posts or papers about various areas. With a large amount of texts, new techniques are needed to analise them quickly and profitably. Text Mining (TM) is a technological discipline that addresses the solution of these problems.

With this powerful technology it is possible to analise a huge amount of texts, in order to obtain useful information. However, most of the current TM approaches do not take advantage of the structure of the texts. A text is seen as a bag of words, a set of unrelated words. This causes the TM algorithms to produce, sometimes, far from optimal quality of the information obtained.

There are different approaches of TM and several steps of pre-processing that can be applied to full text classification, but which methods that can get better results? Which steps must be applied so the final result is more complete?

Nowadays, there is available in the Web, full texts collections. Processing such collections requires much more computational power. In this thesis we investigate if a subset of sections from a text can have similar results in text classification tasks as the full text.

This dissertation concluded that TM algorithms, when applied to small sections of documents, achieve results similar to those obtained by applying the same algorithms to full documents.

Conteúdo

1 Introdução **1**
- 1.1 Contexto . . . 1
- 1.2 Problema . . . 1
- 1.3 Motivação e Objetivos . . . 2
- 1.4 Estrutura da Dissertação . . . 2

2 *Data Mining* e *Text Mining* **3**
- 2.1 *Data Mining* . . . 3
- 2.2 *Text Mining* . . . 12
- 2.3 Ferramentas . . . 15
- 2.4 Sumário . . . 16

3 Classificação de Textos Completos **19**
- 3.1 Arquitetura do Sistema . . . 19
- 3.2 Implementação do Sistema . . . 21
- 3.3 Sumário . . . 24

4 Um Caso de Estudo **25**
- 4.1 Conjunto de Dados . . . 25
- 4.2 Pré-processamento Realizado . . . 26
- 4.3 Classificação dos textos . . . 28
- 4.4 Resultados . . . 28
- 4.5 Sumário . . . 29

5 Conclusões **31**

Referências **33**

CONTEÚDO

Lista de Figuras

2.1 Hiperplano que separa duas classes 4
2.2 Matriz *Error Rate* 10

3.1 Três etapas de processamento do documento 19

4.1 *Script* que lida com a base de dados 25
4.2 Fluxo de pré-processamento dos documentos 27

LISTA DE FIGURAS

Lista de Tabelas

3.1 *Scripts* que executam algoritmos de TM 22
3.2 *Scripts* que calculam o valor de cada palavra 23
3.3 *Script* que cria o ficheiro ARFF 23
3.4 *Scripts* que correm todos os módulos do sistema 24

4.1 Número de documentos relevantes e não relevantes no *dataset* 26
4.2 Distribuição das classes pelos conjuntos de treino e teste no *corpus* de 11 880 documentos. 26
4.3 conjuntos de subsecções utilizadas no estudo. 28
4.4 Resultados obtidos pelos diferentes algoritmos no conjunto *DS1*. 28
4.5 Resultados obtidos pelos diferentes algoritmos no conjunto *DS2*. 29
4.6 Resultados obtidos pelos diferentes algoritmos no conjunto *DS3*. 29
4.7 Resultados obtidos pelos diferentes algoritmos no conjunto *DS4*. 29

LISTA DE TABELAS

Abreviaturas e Símbolos

AL	Apache Lucene
API	Application Programming Interface
ARFF	Attribute-Relation File Format
DM	Data Mining
DNA	DeoxyriboNucleic Acid
DS1	Data Set 1
DS2	Data Set 2
DS3	Data Set 3
DS4	Data Set 4
GM	Graph Mining
HTML	HyperText Markup Language
KDD	Knowledge Discovery in Database
KNN	K Nearest Neighbours
NBC	Naive Bayes Classifier
NER	Named Entity Recognition
NLTK	Natural Language ToolKit
PDF	Portable Document Format
PMID	PubMed-Index for MEDLINE
RM	RapidMiner
RNA	RiboNucleic Acid
SVM	Support Vector Machine
TF-IDF	Term Frequency–Inverse Document Frequency
TM	Text Mining
TREC	Text REtrieval Conference
XML	eXtensible Markup Language

Capítulo 1

Introdução

1.1 Contexto

Vivemos numa época na qual existem diversas áreas, tanto científicas como comerciais, em que é fundamental estar a par dos conhecimentos mais recentes na área. A *Web* possui uma quantidade enorme de informação que está à disposição de todos, incluindo repositórios de documentos científicos (por exemplo a MEDLINE) para os quais, até há bem pouco tempo, tínhamos acesso apenas ao título e resumo do documento. Hoje em dia existe uma maior quantidade de textos completos disponíveis para consulta e análise que são públicos. Com esta disponibilidade de informação é necessária uma ferramenta que nos ajude a identificar quais os documentos relevantes dentro de um determinado tópico. O volume e complexidade das análise de textos completos requer uma reavaliação das técnicas existentes.

Text Mining é um conjunto de métodos e algoritmos que permitem (entre outras tarefas) classificar textos. É neste sentido que o TM pode ser uma solução para recolher documentos relevantes para um determinado tópico, pois é um método automático que consegue identificar conceitos importantes num texto.

1.2 Problema

O TM dispões hoje de um conjunto de técnicas poderosas mas que ainda têm limitações. As abordagens atuais de TM não tiram (ou tiram pouco) partido da estrutura e relação existente nos textos. Os algoritmos de TM tratam os textos como um saco de palavras (*bag of words*) o que pode tornar os algoritmos menos eficazes. Existem técnicas que são capazes de codificar e manipular informação relacional o que pode ajudar a melhorar o desempenho dos algoritmos de TM.

1.3 Motivação e Objetivos

Sendo o TM uma ferramenta tão útil na atualidade, é importante procurar melhorar o seu desempenho. Por esta razão, esta dissertação pretende estudar o estado-da-arte de DM e de TM e criar um sistema flexível que permita realizar várias tarefas de pré-processamento de TM, de forma a testar quais as mais eficientes e que obtêm melhores resultados.

Este trabalho pretende desenvolver uma aplicação que utilize métodos tradicionais de TM e avalie até que ponto as abordagens tradicionais são aplicáveis a textos completos. Processar textos completos implica a utilização de recursos computacionais mais poderosos. Nesta dissertação pretendemos também avaliar se a utilização de um número reduzido de secções dos textos permite construir classificadores com desempenho semelhante aos classificadores construídos com os textos completos.

1.4 Estrutura da Dissertação

Para além da introdução, esta tese tem mais mais quatro capítulos. No Capítulo 2 é estudado o estado-da-arte relativo a DM e TM. Além disso, são descritas as ferramentas utilizadas na realização da tese. No Capítulo 3 é descrita a proposta de trabalho realizado no âmbito desta dissertação. No Capítulo 4 apresentamos os resultados obtidos no caso de estudo através do sistema desenvolvido no âmbito desta dissertação. No Capítulo 5 são indicadas as conclusões retiradas do trabalho efetuado nesta dissertação.

Capítulo 2

Data Mining e *Text Mining*

Neste capítulo introduzimos os conceitos básicos e descrevemos as ferramentas relevantes para o trabalho de tese. São abordadas as áreas de *Text Mining* (TM) e *Data Mining* (DM). Por fim descrevemos as ferramentas relevantes para o nosso estudo.

2.1 *Data Mining*

Data Mining [1] é uma abordagem à análise de dados que aborda várias tarefas possíveis: classificação, regressão, Descoberta de Regras de Associação, Clustering, entre outras. No nosso trabalho estamos particularmente interessados em tarefas de classificação ou Clustering (utilizado como processo de classificação).

Classificação

Existem vários algoritmos que vamos abordar e que podem ser utilizados no treino de um classificador, tais como: *Support Vector Machine*, *Naive Bayes Classier*, *K-Nearest Neighbor*, Algoritmo de Rocchio, *Decision Trees* e *Random Forest*. Estes algoritmos estão descritos em [dPVOG13].

Support Vector Machines

Support Vector Machine (SVM) é uma técnica de *machine learning* para classificação de textos. Este modelo foi proposto por [Vap99]. Em SVM, os documentos são representados como pontos num espaço vetorial, onde as dimensões são as características escolhidas. A ideia básica do SVM é encontrar um hiperplano ótimo com base no treino de vetores de documentos, como mostra a Figura 2.1, de forma a separar duas classes de pontos com a maior margem possível dos

[1] Em rigor a designação deveria ser *KDD – Knowledge Discovery in Data bases*

dados pré-processados. Depois de determinado o hiperplano, pode ser utilizado para classificar os pontos em duas classes com base no lado do hiperplano em que estes se encontram.

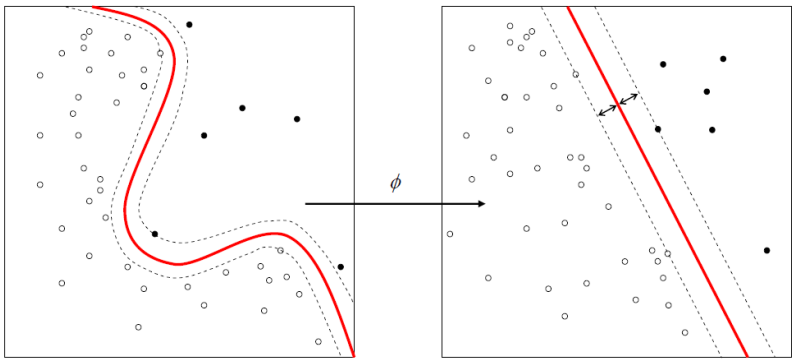

Figura 2.1: Hiperplano que separa duas classes

Quando é aplicado a classificação de textos, uma das características básicas é a frequência das palavras, a ideia é que os documentos são constituídos por palavras e a ocorrência dessas palavras são pistas para a classificação do documento.

Este método traz vantagens e desvantagens. Nas vantagens podemos ter em conta:

- Lidar com problemas complexos do mundo real;
- SVM tem um bom desempenho em conjuntos de dados com muitos atributos, mesmo com poucos casos para treinar o modelo.

As desvantagens são:

- Só considera duas classes;
- Sensível a ruído, um número relativamente pequeno de exemplos mal classificados pode prejudicar o desempenho drasticamente.

Em geral, SVM consegue resultados com elevada precisão, quando comparado a outros métodos.

Naive Bayes Classier

Naive Bayes Classier (NBC) é um classificador de aprendizagem supervisionada probabilístico baseado no Teorema de Bayes [dPVOG13]. *Naive Bayes* assume que a existência de um tipo de características não depende da existência de outras características. O algoritmo de *Naive Bayes* precisa de um conjunto de documentos de treino já classificados para construir um modelo de aprendizagem. Os novos documentos serão classificados com base na probabilidade de pertencerem ou não a uma classe predefinida. O algoritmo NBC calcula $P(C_k|D)$, a probabilidade que um documento D pertencer à classe C_k. De acordo com o Teorema de *Bayes*: $P(C_k|D) = \frac{P(C_k)*P(D|C_K)}{P(D)}$

$P(C_k|D) = \prod_i P(W_i|C_k)$

As vantagens do NBC são:

- Fundamentação teórica explícita (com base no Teorema de Bayes);
- Relativamente eficaz;
- Implementação simples e fácil;
- Treino e classificação rápidos;
- Robusto contra ruído.

As suas desvantagens são:

- Pressuposto da independência das características é errado para classificação de textos;
- Tem um desempenho pior do que outros métodos;
- Em alguns *datasets* falha completamente.

K-Nearest Neighbor

O classificador *K-Nearest Neighbor* (KNN) é um algoritmo de aprendizagem supervisionada. KNN classifica um novo exemplo comparando-o a todos os exemplos anteriores. A classificação dos k casos anteriores mais similares é usada para prever a classificação do novo exemplo. O algoritmo KNN [dPVOG13]:

- Dado um documento de teste:
    1. Encontrar os K vizinhos mais próximos nos documentos de treino;
    2. Calcular e ordenar as pontuações de cada categoria para cada candidato;
    3. Fazer o *threshold* destas pontuações.

Regra de decisão:
$Y(\overrightarrow{x}, C_i) = sign \sum d_i \in KNN(sim(\overrightarrow{x}, \overrightarrow{d_i}) Y(\overrightarrow{d_i}, c_j - b_j))$
$Y(d_i, c_j) \in 0, 1$
$sim(\overrightarrow{x}, \overrightarrow{d_i})$ semelhança entre o documento de teste x e o documento de treino d_i
b_j: Categoria - *Threshold* específico
Podemos apontar algumas vantagens e desvantagens do KNN. A vantagens são:

- Simples e indutivo;
- Bastante utilizado;
- Está entre as ferramentas com melhor desempenho em classificação de texto;
- Fácil implementação.

As desvantagens são:

- Aproximação heurística;
- É difícil determinar o número de vizinhos k
- É computacionalmente pesado com *datasets* grandes.

Algoritmo de Rocchio

O algoritmo de Rocchio utiliza vetores ponderados *tf-idf* para representar documentos, e constrói um vetor para cada categoria juntando os vetores de documentos de treino para cada categoria. Os documentos de teste são atribuídos à categoria que tem o vetor mais próximo, com base similaridade dos vetores [dPVOG13]. O classificador de Rocchio tem um número fixo de categorias que são conhecidas previamente. Cada documento é atribuído a exatamente uma das categorias.

O classificador de Rocchio é baseado em *tf-idf*. Foi proposto inicialmente como um algoritmo de relevância de *feedback*, mas passou a ser utilizado na área da classificação de textos. Tanto as palavras como os documentos são representados por vetores e, quanto mais perto um documento $\overrightarrow{d_j}$ é da palavra de vetor $\overrightarrow{w_j}$, maior a semelhança entre o documento e a palavra.

Um documento d é representado por $d = (d_1, d_2, d_3, ..., d_r)$, onde cada dimensão é a probabilidade de um termo t_i do documento multiplicada pelo inverso da frequência do termo em todos os documentos $IDF(t_i)$. $d_i = P(t_i|D) * IDF(t_i)$.

O inverso da frequência do termo é definido por um logaritmo do inverso da probabilidade do termo em todos os documentos D, $IDF(t_i) = log\frac{1}{P(t_i|D)}$.

A ideia fundamental do algoritmo de Rocchio é construir uma *optimal query* para que a diferença entre a média de pontuação de um documento relevante e a média de pontuação de um documento irrelevante sejam maximizadas.

A técnica de Rocchio constrói para cada categoria um único documento protótipo. O perfil da categoria consiste numa lista ponderada de termos formada pela distribuição de palavras pertencentes à categoria. Para decidir a qual categoria o novo documento pertence, a sua distribuição de palavras é comparada com os documentos protótipo de cada categoria. Quando a semelhança é elevada, o novo documento é associado à categoria.

Decision Trees

Aprendizagem com árvores de decisão (*Decision Trees*) é um dos algoritmos mais bem sucedidos por vários fatores: simplicidade, compreensibilidade e capacidade de lidar com tipos diversos de dados [dPVOG13]. O principal objetivo do algoritmo é construir uma árvore com testes que diferenciem documentos de classes diferentes. Uma decisão de classificação é uma sequência de testes que resultam na atribuição de uma categoria correspondente a um nó da árvore. A árvore de decisão é construida através da análise dos conjuntos de teste para os quais as classes são identificadas. De seguida, são aplicadas para classificar documentos novos. Se treinadas com dados de qualidade, as árvores de decisão podem fazer classificações precisas.

Podemos apontar algumas vantagens e desvantagens das árvores de decisão. As vantagens são:

- Fácil interpretação;
- Fácil implementação;
- Treino e classificação rápidos.

As desvantagens são:

- Analisa um atributo de cada vez, portanto não há hipótese dedetetar iteração entre variaveis.

Existem várias metedologias nos algoritmos de *decision trees*, tais como:

- ID3

  Iterative Dichotomiser 3 (ID3) é um algoritmo inventado por Ross Quinlan [Qui86]. O algoritmo ID3 começa com um conjunto inicial S como o nó raíz. Em cada iteração do algoritmo:

  1. Calcula-se a entropia de cada atributo de S;
  2. Divide-se S em subconjuntos usando os atributos em que a entropia á mínima;
  3. Cria-se um nó da árvore que contenha esse(s) atributo(s);
  4. Repete-se a sequência com os atributos restantes.

  Entropia é a quantidade de incerteza que existe no *dataset (S)*.

  $H(S) = -\sum_{x \in X} p(x) log_2 p(x)$

  Tal que:

  - S é o *dataset* atual para o qual estamos a calcular a entropia;
  - X é o conjunto de categorias em S
  - $p(x)$ a proporção do número de elementos da classe x para o número de elementos em S.

  Quando $H(S) = 0$, o conjunto S está perfeitamente classificado (todos os elementos de S são da mesma categoria). Em ID3, a entropia é calculada para cada atributo restante. O atributo com o menor entropia é usado para dividir o conjunto S.

- C4.5

  C4.5 é um algoritmo utilizado para gerar *decision trees*, também desenvolvido por Ross Quinlan [Qui93]. Este algoritmo é uma extensão do algoritmo anterior. C4.5 cria árvores de decisão da mesma forma que o algoritmo ID3, utilizando a entropia. O conjunto de treino é um conjunto $S = s_1, s_2, s_3, ..$ de exemplos já classificados. Cada exemplo s_i consiste num vector de p dimensões $(x_{1,i}, x_{2,i}, x_{3,i}, ..., x_{p,i})$, no qual x_j o peso dos atributos da amosta e da categoria que caracteriza s_i. Este algoritmo tem os seguintes casos base:

- Todos os exemplos no vetor pertencem à mesma categoria. Quando isto acontece, o algoritmo cria um nó que escolhe essa categoria.
- Nenhum dos atributos oferece informação adicional. Neste caso, C4.5 cria um nó de decisão usando o valor esperado da categoria.
- Instância da categoria anteriormente não visível é encontrada. Novamente, C4.5 cria um nó de decisão usando o valor esperado da categoria.

O algoritmo tem os seguintes passos:

1. Verificar se algum caso base existe;
2. Para cada atributo *a*, encontrar a razão do ganho da informação normalizado pelo particionamento em *a*;
3. Seja *a_best* o atributo com maior ganho da informação normalizado;
4. Criar um nó de decisão que divide o conjunto de dados em *a_best*;
5. Repetir nos subconjuntos obtidos através da divisão em *a_best*.

Random Forest

O classificador *Random Forest* foi desenvolvido por Breiman [Bre01]. Este algoritmo combina árvores de decisão individuais em conjuntos. Este algoritmo cria árvores binárias (cada nó gera apenas dois nós filhos). Cada árvore vota numa determinada classe e a classe mais votada é escolhida como *output* do classificador. Cada árvore tem o mesmo peso nas votações, ou seja, não existem árvores cujo voto valha mais que os outros. Este algoritmo consegue ser bastante eficiente, principalmente em *datasets* de grande dimensão, mas necessita de mais memória do que a maioria dos algoritmos.

Algumas das vantagens do algoritmo *Random Forest*, segundo [DUAdA06], são:

- Pode ser utilizado quando existem muitas variáveis;
- Pode ser utilizado em problemas de duas ou mais categorias/classes;
- Consegue obter bons resultados e um bom desempenho, mesmo que exista muito ruído nas variáveis.

As desvantagens são:

- Necessita de muita memória comparado a outros algoritmos.

Métricas e Métodos de avaliação

Existem métodos de avaliação que são utilizados na área do DM tais como: *Hold-Out*, *Cross Validation*, *Leave One Out* e *Bootstrap* [JaJ10].

Hold-Out

O método de *Holdout* divide os dados originais em dois conjuntos: treino e teste. Este método de avaliação permite assim testar o modelo preditivo e avaliar o seu desempenho. Tanto o conjunto de treino como o conjunto de teste devem ser grandes e conter muitos exemplos. Isto pode tornar-se um problema pois nem sempre existem dados suficientes.

Cross Validation

Quando apenas uma quantidade limitada de dados está disponível para conseguir uma estimativa imparcial do desempenho do modelo utiliza-se *Cross Validation*. No *Cross Validation* divide-se o *dataset* em k conjuntos de igual tamanho. De seguida, criam-se k modelos utilizando os sub-conjuntos criados anteriormente, mas retira-se um sub-conjunto a cada modelo. Desta forma cada modelo terá $k-1$ conjuntos de dados. Os modelos criados são utilizados como conjunto de teste.

Leave One Out

O método *Leave One Out* é semelhante ao *Cross Validation*. O *dataset* original também é dividido em k conjuntos, mas com a particularidade de o valor de k ser igual ao tamanho da amostra do *dataset* original. Neste método os modelos criados deixam um elemento do *dataset* original de fora. Este método torna-se muito pesado computacionalmente.

Bootstrap

No *Bootstrap* criam-se modelos de treino a partir do *dataset* original. Estes modelos contém n elementos, sendo n o número de amostras do *dataset* original. Os modelos criados contém elementos repetidos , por exemplo, se o *dataset* original tiver os elementos:

120 123 103 107 102 100

Alguns possíveis conjuntos criados são:

123 103 120 120 102 102

100 100 100 100 100 100

120 123 103 107 102 100

Para teste, utilizam-se os elementos do *dataset* original que não ocorrem no novo conjunto de treino.

As métricas de avaliação que vão ser descritas são: *Accuracy*, *Error Rate*, *Precision*, *Recall* e *F-Measure*.

Accuracy

Accuracy é uma métrica que avalia o desempenho de um classificador. Para calcular a *Accuracy* de um classificador basta dividir o número de previsões corretas pelo numero total de previsões que o classificador efetuou.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ of\ all\ cases\ to\ be\ predicted}$$

Accuracy não é uma métrica viável quando o número de amostras de diferentes categorias varia muito. Por exemplo, se o *dataset* tiver 95 gatos e 5 cães, o classificador facilmente classificará todas as amostras como gatos. Neste caso a *Accuracy* será de 95% sendo que na realidade o classificador tem 100% racio de classificação para gatos e 0% racio de classificação para cães.

Error Rate

Error Rate é uma métrica para problemas de classificação binários. Esta métrica utiliza uma matriz como a da Figura 2.2

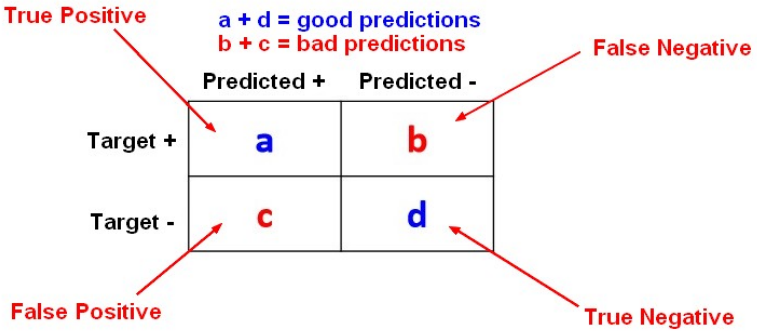

Figura 2.2: Matriz *Error Rate*

Nesta matriz podem ser observados as previsões corretas e incorretas. Além disso separa as previsões positivas das previsões negativas. Exitem três "*Rates*"que podem ser calculados através da informação fornecida pela matriz: *False positive rate*, *False negative rate* e *True positive rate*.

- *False positive rate*

  A fração dos exemplos negativos que são classificados como positivos.

  $$False\ positive\ rate = \frac{False\ Positive}{False\ Positive + True\ Negative}$$

  Sendo *False Positive* previsões que deveriam ser negativas mas foram classificadas como positivas e *True Negative* previsões que foram corretamente classificadas como negativas.

- *False negative rate*

A fração de exemplos positivos que são classificados como negativos.

$$\textit{False negative rate} = \frac{\textit{False Negative}}{\textit{True Positive} + \textit{False Negative}}$$

Sendo *False Negative* previsões que deveriam ser positivas mas foram classificadas como negativas e *True Positive* previsões que foram corretamente classificadas como positivas.

- *True positive rate*

  A fração de exemplos positivos corretamente classificados.

$$\textit{True positive rate} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Negative}}$$

  Sendo *True positive* previsões que foram corretamente classificadas como positivas e *False negative* previsões que deveriam ser positivas mas foram classificadas como negativas.

Com estas métricas podemos perceber melhor a qualidade do desempenho do nosso classificador. Mas apenas podem ser aplicadas a classificadores binários.

Precision

Precision é a fração de documentos retornados que são relevantes.

$$\textit{Precision} = \frac{|\{\textit{relevant documents}\} \bigcap \{\textit{retrieved documents}\}|}{|\{\textit{retrieved documents}\}|}$$

Considerando uma categoria, um documento relevante (*relevant documents*) é um documento que pertence a essa categoria, enquanto um documento retornado (*retrieved documents*) é um documento que o classificador considerou relevante. A formula consiste na divisão de o número de todos os documentos que o classificador classificou corretamente como relevantes pelo número de todos os documentos que o classificador classificou como relevantes.

Recall

Recall é a fração de documentos que são relevantes que são retornados corretamente.

$$\textit{Precision} = \frac{|\{\textit{relevant documents}\} \bigcap \{\textit{retrieved documents}\}|}{|\{\textit{relevant documents}\}|}$$

O métrica de *Recall* é semelhante ao *Precision* mas em vez de considerar todos os documentos retornados pelo classificador, considera todos os documentos relevantes na categoria em causa.

F-Measure

F-Measure é uma métrica que combina as métricas de *Precision* e *Recall*. *F-Measure* é calculado através da fórmula:

$$F = 2 \cdot \frac{\textit{Precision} \cdot \textit{Recall}}{\textit{Precision} + \textit{Recall}}$$

Esta medida é aproximadamente a média das métricas *Precision* e *Recall* quando os dois valores são próximos. Sendo *F-Measure* uma métrica que combina as duas métricas, *Precision* e *Recall*, esta torna-se uma métrica mais completa e que permite avaliar um classificador com maior precisão.

Estes foram alguns métodos e métricas que podem ser utilizadas testar e avaliar o desempenho de um classificador.

2.2 *Text Mining*

O objetivo do TM é descobrir automaticamente informação, inicialmente desconhecida, através da extração de conhecimento de uma quantidade grande de diferentes recursos textuais que se encontram desorganizados estruturalmente [CG-15]. O facto da informação textual não se encontrar previamente estruturada, torna os algoritmos de TM, por vezes, muito pesados e com tempos de execução elevados.

Processo do *Text Mining*

O processo de TM pode ser separado nos seguintes passos [CG-15] :

- *Data Selection*;

  É necessário escolher um conjunto de dados de treino que seja robusto, completo e que abranja todos os tópicos que queremos detetar nos futuros textos avaliados. Em tarefas de classificação os textos têm que ser classificados por especialistas.

- *Term Extraction and Filtering*;

  A extração e filtragem dos termos chave é possivelmente o passo mais importante dos algoritmos de TM pois é difícil entender quais das palavras encontradas são palavras chave (palavras importantes). Se retirarmos apenas as palavras que aparecem mais vezes vamos obter informação inútil como artigos, preposições e conjunções. É por esta razão que a filtragem é tão importante em TM para separar informação útil e importante da informação inútil.

- *Data Clustering / Classification*;

  Tendo já pré-processado o *dataset*, podem-se definir duas tarefas diferentes: *clustering* ou classificação. O *clustering* é geralmente uma tarefa utilizada para organizar a coleção de textos. Nesta tarefa os textos não precisam de ser previamente classificados. Numa tarefa de classificação os textos é construido um modelo de classificação (um classificador) e esse modelo serve para classificar novos textos.

- Mapeamento e Visualização;

  Neste passo é essencial que os dados obtidos possam ser visualizados de forma clara para que seja fácil para o utilizador identificar a informação mais relevante.

- Interpretação dos resultados;

  Na secção de interpretação de resultados deve conseguir perceber a informação que é retirada da aplicação do TM e qual o resultado final da aplicação do algoritmo. Neste passo deve-se conseguir separar os textos analisados por tema, dependendo da robustez do algoritmo utilizado.

Processamento de documentos

O TM pode ser aplicado aos mais diversos tipos de documentos, com extensões diferentes. Podem ser documentos *HTML*, *PDF*, *XML*, *.doc* ou um simples *.txt*. Para extrair informação de uma grande quantidade de documentos é necessário processá-los previamente. Existem diversas técnicas para processar documentos, como as descritas em [CG-15]. Estas técnicas são, normalmente, utilizadas na etapa *Term Extraction and Filtering* do algoritmo TM e incluem os seguintes passos:

- *Tokenization and removing unwanted characters*;

  Este primeiro passo tem como objetivo retirar caracteres desnecessários, como por exemplo *tags* do HTML/XML ou pontuação que não oferece informação adicional (. , ; ! ? [ ] etc) de forma a que todas as palavras fiquem separadas por espaços. Este é um processo complexo pois é necessário entender qual o domínio do texto e quais os caracteres que podem conter informação importante. Por exemplo, no tema da biologia pode ser importante manter os caracteres + e - pois estes podem caracterizar um tipo sanguíneo. Em textos literários a pontuação pode também ser relevante.

- *Stop Words Removal*;

  Neste passo são removidas as palavras sem significado nem interesse, palavras que não oferecem informação adicional, tais como artigos, preposições e conjunções. Este passo além de remover palavras desnecessárias também reduz significativamente o número de termos que o algoritmo de TM tem de analisar.

- *Stemming*;

  Stemming é um processo que reduz palavras derivadas, transformando-as na sua palavra raiz. Se aplicarmos *stemming* a palavras como: computador, computar e computação estas palavras serão reduzidas a "computa", pois é a sua raiz. Este processo consegue reduzir plurais e conjugações verbais e, assim, o modelo de aprendizagem consegue classificar um documento corretamente, reduzindo também o número de palavras avaliadas. Este processo tem uma desvantagem. Por vezes, palavras que têm a mesma raiz, têm significados distintos,

por exemplo, os termos ingleses *desert* e *dessert* são palavras cuja raiz é "des", mas têm significados completamente diferentes. Apesar de *stemming* não ser 100% eficaz, continua a ser bastante utilizado na classificação de textos pois ajuda a melhorar consideravelmente os classificadores [Seb02]. O algoritmo de *Porter* [Por80] é um dos mais conhecidos usos de *stemming*.

- *Named Entity Recognition*;

  Named Entity Recognition é uma tarefa de identificação de termos que mencionam uma entidade conhecida. Esta entidade pode ser, por exemplo, uma pessoa, um local, uma organização.

- *Synonyms Handling*;

  Synonyms Handling é o processo de substituir palavras do texto por sinónimos de modo que todos as palavras que tenham o mesmo significado sejam substituídas pela mesma palavra. Este passo permite reduzir significativamente o número de termos no *dataset* sem alterar o significado do texto.

- *Word Validation*;

  Esta etapa tem como objetivo validar as palavras encontradas pelo algoritmo comparando-as às palavras existentes num dicionário. Existem dicionários disponíveis *online* com termos comuns, principalmente em inglês, e outros dicionários com termos mais específicos, por exemplo no domínio da medicina ou da biologia.

- *Pruning*;

  Pruning ou, em português, poda refere-se ao ato de remover ramos menos promissores de uma árvore de forma a que ela seja mais rentável. No TM também existem termos menos promissores que pretendemos remover à partida. A lei de *Zipf* [Pow98] é uma medida linguística que mede a frequência das palavras num documento. De acordo com esta lei, uma palavra que apareça poucas vezes num documento é, normalmente, estatisticamente insignificante. Portanto, a probabilidade desta palavra aparecer num texto do mesmo tema é também bastante baixa. *Pruning* é útil na fase de pré-processamento do texto removendo a maioria das palavras de frequência baixa [Hoo11]. Na fase de classificação, o *pruning* reduz o espaço de características textuais, resultando um modelo de classificação menor e um melhor desempenho nos testes dos *datasets* devido à irrelevância que as palavras de baixa frequência apresentam na fase de classificação do texto [MN98], [Joa98].

Com estes processos podemos criar um algoritmo TM eficaz e com um nível de desempenho bastante aceitável. Haverá formas de tornar este método de TM ainda mais rápido e eficaz?

2.3 Ferramentas

Para o desenvolvimento de protótipos para testar algoritmos de TM existem algumas ferramentas que podem ser úteis e permitem poupar muito tempo pois já possuem funções especificas de TM.

Ferramentas específicas de *Data Mining*

- *RapidMiner*

  RapidMiner (RM) é uma plataforma de análise preditiva, *open source*, que permite treinar classificadores e classificar novos conjuntos de dados [RM].

  RM inclui funções e ferramentas preparadas para a análise de documentos. RM é uma ferramenta bastante gráfica que dá ao utilizador a possibilidade de implementar algoritmos de DM sem escrever uma linha de código.

- R

  A linguagem de programação R foi criada principalmente para trabalhar na área de estatística computacional [R]. R possui uma biblioteca chamada TM [FH15], [FHM08] que contém funcionalidades de *text mining*.

- WEKA

  WEKA é um software, desenvolvido em *Java*, específico para criar algoritmos de DM [WEK]. Além disso, o WEKA tem um conjunto de algoritmos de *machine learning* especializados para tarefas de TM. O WEKA pode ser executado diretamente ou chamado através de sua *Application Programming Interface* (API) em *Java*.

  WEKA utiliza ficheiros ARFF para a classificação dos textos. O ficheiro ARFF possui duas partes: Cabeçalho e Corpo [ARF]. No Cabeçalho é definido o conteúdo que o resto de ficheiro terá, quais os atributos que cada linha do ARFF terá e que tipo de valores têm. O Corpo é constituído por várias linhas que representam uma instância com os atributos definidos no Cabeçalho.

Ferramentas específicas de *Text Mining*

- Apache Lucene

  Apache Lucene (AL) é um motor de busca, *open source*, desenvolvido em *Java* que oferece ao utilizador uma plataforma para pesquisa de termos num texto com suporte para multiplataforma [luc].

  Existe uma documentação *Java Docs* do AL que facilita a utilização da ferramenta. Além disso, o *website* oficial disponibiliza um tutorial. AL é uma poderosa ferramenta para desenvolver *software* e algoritmos de TM.

- Gate

  Gate é um *software open source* capaz de resolver quase todos os problemas de processamento de texto [Gat]. Tem uma comunidade formada por pessoas diferentes: *developers*, alunos, professores e cientistas.

- Stanford Parser

  Um *parser* de linguagem natural é um programa que trata da estrutura gramatical das frases, por exemplo, quais os grupos de palavras que aparecem juntas e que palavras são o sujeito e o predicado do verbo da frase [Sta]. Stanford Parser é um *parser* probabilístico que usa o conhecimento adquirido através da análise textual para perceber a organização gramatical das frases.

Outra Ferramentas

- Python

  Python [Pyt] é uma linguagem poderosa e rápida que permite criar código facilemente. Pode ser instalado em qualquer máquina, *open source* e possuí várias bibliotecas uteís tal como o NLTK (Natural Language Toolkit).

- NLTK

  NLTK [NLT] é uma plataforma desenvolvida em python que trabalha com linguagem natural. Oferece interfaces de uso fácil e um conjunto de bibliotecas de processamento de texto. É uma ferramenta que facilita o pré-processamento de texto devido às funções que contém. O NLTK possuí dois *packages* que facilitam no desenvolvimento de algoritmos de *text mining*: Stem e WordNET.

  O Stem é um *package* do NLTK que permite facilmente fazer o *stemming* das palavras de um texto. Este *package* contém um dicionário de palavras e das suas raízes, o que torna a tarefa do *stemming* muito mais simples.

  O WordNET é um *package* que contém vários dicionários (vários idiomas) que permitem validar palavras e obter sinónimos. Com um dicionário de sinónimos é possível encontrar palavras, num documento, que sejam sinónimos e convertê-las numa mesma palavra.

  Assim, o NLTK torna-se uma ferramenta muito poderosa e que pode ser uma grande ajuda na criação de alguns algoritmos de *text mining*.

2.4 Sumário

Esta pesquisa permite concluir que já existe bastante trabalho realizado na área do TM e muitas ferramentas que podem facilitar e acelerar o desenvolvimento deste trabalho. Para a realização

deste trabalho, utilizamos python, por ser uma linguagem acessível e facilmente testável, e o WEKA, devido aos algoritmos de DM que tem implementados.

Capítulo 3

Classificação de Textos Completos

Neste capítulo é descrito o trabalho realizado no âmbito desta dissertação. O objetivo é criar um sistema de *text mining* versátil, flexivel e que possa ser reutilizado.

3.1 Arquitetura do Sistema

O sistema foi desenvolvido em python, procurando atribuir cada tarefa a um *script* (módulo) diferente. Estas tarefas podem ser divididas em 3 partes, tal como mostra a Figura 3.1:

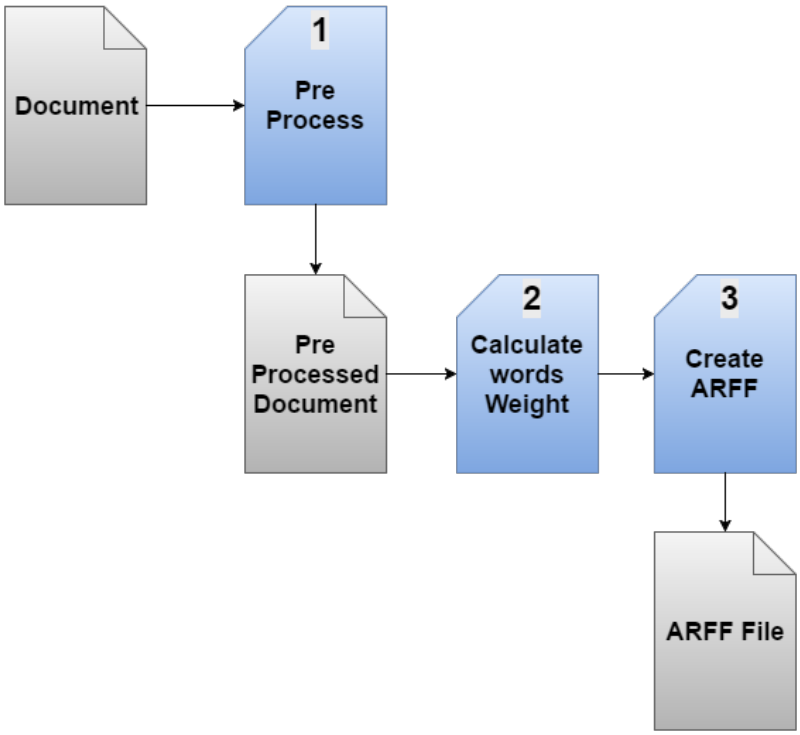

Figura 3.1: Três etapas de processamento do documento

1. Obtenção e pré-processamento dos textos;
2. Cálculo do peso de cada palavra nos textos;
3. Criação do ficheiro ARFF para a posterior classificação;

Foi fornecido um conjunto de documentos (*TREC Genomics 2005 biomedical text corpus*). Estes documentos estão classificados como relevantes ou não relevantes em duas categorias: *Allele – alleles of mutant phenotypes and GO – gene ontology annotation* [HV09].

Foi criado um *script* que retorna uma secção de qualquer documento escolhido pelo utilizador. Com isto, é possível, por exemplo, retirar apenas o resumo de um documento ou juntar a introdução e a conclusão e passar essa informação para um ficheiro de texto.

Para concluir este primeiro passo, foram criados vários módulos de forma a que cada um execute uma parte do pré-processamento. A ideia é que cada *script* receba (entre outros argumentos) um ficheiro de texto de *input* e retorne um ficheiro de texto de *output*. Este formato implica que cada passo do pré-processamento necessite de ler o ficheiro completo, fazendo com que o sistema percorra o ficheiro de texto várias vezes. Isto torna-se um processo menos eficiente, mas mais flexível pois pode-se correr os diferentes módulos pela ordem que quisermos e, dessa forma, experimentar diferentes métodos de pré-processamento. Os vários módulos criados para pré-processamento são:

- *Character Removal*;
- *Stop-Words Removal*
- Validação de Palavras
- *Named Entity Recognition (NER)*
- *Stemming*
- Sinónimos

Estes módulos permitem pré-processar os documentos de forma a obter melhores resultados na contagem e análise das palavras dos textos.

Na segunda fase deste processo pretende-se calcular o peso de cada palavra no texto. Para isso, verifica-se a frequência de cada palavra em cada texto, usando o *dictionary* do python. Um *dictionary* em python é um tipo de dados que pussui chaves (*keys*) que mapeiam para um valor de qualquer tipo (*string*, *int*, *etc.*). Sendo que uma *key* também pode ser de qualquer valor, pode-se ter um *string key* a apontar para um inteiro. Desta forma uma palavra aponta para o número de vezes que esta aparece no texto. Depois de contadas as palavras em cada texto, são guardadas numa base de dados para futuro cálculo do valor do *tf-idf*.

Term Frequency–Inverse Document Frequency (Tf-idf) é um cálculo que mostra o valor que cada palavra tem em cada texto. A fórmula para calcular o *tf-idf* é a frequência da palavra num texto vezes o inverso da frequência dessa palavra em todos os textos. Desta forma, uma palavra que aparece muitas vezes num texto não terá tanto valor se aparecer igualmente muita vezes nos outros textos. Este método ajuda a distinguir palavras relevantes para o texto de palavras que são comuns.

Foram criados mais dois módulos complementares do módulo de contagem de palavras: um filtro que remove palavras de menor e maior frequência e contagem de *n-gram*. O módulo de filtragem recebe uma percentagem máxima e uma percentagem mínima e todas as palavras cuja percentagem de frequência seja menor que a mínima ou maior que a máxima são removidas da base de dados. O cálculo da percentagem de frequência de uma palavra é feito para cada texto em que a palavra aparece. Se a palavra tiver uma frequência muito baixa num texto, é removida desse texto. O módulo de contagem dos *n-gram* é um melhoramento do módulo de contagem de palavras. Em vez de contar apenas palavra a palavra, é possível fornecer ao módulo um valor *n* que indica o tamanho dos conjuntos de palavras contados. Por exemplo, quando *n* é 1 a contagem é normal e conta-se palavra por palavra. Caso o valor de *n* seja 2 serão contados grupos de duas palavras. Este método tem como objetivo encontrar excertos de texto frequentes.

Na última fase deste sistema é criado o ficheiro ARFF para que, de seguida, o programa WEKA possa classificar os textos. Foi criado um módulo em python que cria o ARFF utilizando os tf-idf calculados. O *script* recebe um lista de números de identificação dos textos que vai adicionar ao ARFF e cada linha do ficheiro possui um número de identificação do texto, os tf-idf das palavras que existem em todos os textos da lista (valores de tf-idf inexistentes passam a ?) e se o texto é relevante ou irrelevante para o tópico. O cabeçalho do ficheiro ARFF tema o seguinte formato:

```
@RELATION NOME_INFORMATIVO
@ATTRIBUTE pmid NUMERIC
@ATTRIBUTE word1 NUMERIC
@ATTRIBUTE word2 NUMERIC
@ATTRIBUTE word3 NUMERIC
...
@ATTRIBUTE class {pos,neg}
@DATA
...
```

Desta forma pode-se processar os textos, aplicando os algoritmos pretendidos e obter no final um ficheiro ARFF.

3.2 Implementação do Sistema

O sistema está implementado em python e os documentos estão guardados numa base de dados MySQL. Todos os textos têm um número de identificação denominado de *pmid*. Cada texto está

dividido por secções: *title*, *abstract*, *introduction*, *methods*, *conclusions*, *results* e *body* (Sendo *body* a combinação de *methods*, *conclusions* e *results*). Para obter o texto completo basta concatenar as secções *abstract* e *body* da base de dados. Desta forma, é possível correr os algoritmos de pré-processamento para uma secção específica dos textos. Na experiência, utilizamos dois conjuntos de dados de *TREC Genomics 2005 biomedical text corpus* [HV09] cujos textos (mais de 11 mil) estão classificados como relevantes ou não relevantes. Este conjunto de textos foi usado para duas tarefas de classificação distintas:

- *Allele – alleles of mutant phenotypes*;
- *GO – gene ontology annotation*.

Para treinar o classificador e testa-lo, os textos são divididos em treino e teste, sendo que o conjunto de textos de treino não contêm nenhum texto do conjunto de testes e vice-versa. Na base de dados são também guardadas as tabelas *count* e *tfidf* que contém os valores das frequências e dos *tf-idf* de cada palavra para cada texto examinado.

O sistema retira uma secção escolhida de um número de textos selecionado e corre a lista de algoritmos de forma a obter melhores resultados na fase de classificação dos textos de treino.

Para o utilizador do sistema escolher quantos textos quer processar, quais os algoritmos que quer aplicar ou qual a secção do texto que vai ser utilizada, existe um ficheiro de *settings* que contém toda essa informação e que pode ser alterada pelo utilizador, dependendo do tipo de experiência que queremos efetuar.

No ficheiro de *settings* podem ser também definidas algumas variáveis utilizadas pelos módulos, por exemplo: o ficheiro com as *stop-words* a utilizar, o ficheiro com os caracteres que queremos remover e as percentagem mínima e máxima de frequência de palavras para o filtro.

Os módulos criados para o estudo de algoritmos de TM estão representados na Tabela 3.1:

Tabela 3.1: *Scripts* que executam algoritmos de TM

Name	*Input*	*Output*
Character Remove	Ficheiro de texto de *input* Ficheiro com caracteres a remover	Ficheiro de texto sem caracteres
Stop-Words Remove	Ficheiro de texto de *input* Ficheiro com *stop-words* a remover	Ficheiro de texto sem *stop-words*
Stemming	Ficheiro de texto de *input*	Ficheiro de texto com as palavras alteradas para a sua raíz
NER	Ficheiro de texto de *input*	Ficheiro de texto com entidades reconhecidas
Word Validation	Ficheiro de texto de *input*	Ficheiro de texto sem palavras inválidas
Synonyms	Ficheiro de texto de input	Ficheiro de texto com palavras alteradas para os seus sinónimos

Na Tabela 3.2 pode-se ver os módulos utilizados para a segunda fase do processo. Depois de os textos estarem pré-processados, os módulos de *count* e *tf-idf* calculam o valor que cada palavra tem num texto.

Tabela 3.2: *Scripts* que calculam o valor de cada palavra

Name	*Input*	*Output*
Count	Ficheiro com os pmid dos textos que queremos analisar Número de *n-gram* a analisar	Guarda na tabela *count* o valor das frequências de cada palavra (ou conjuntos de palavras) em cada texto
Filter	Percentagem mínima de frequência das palavras Percentagem máxima de frequência das palavras	Altera a tabela *count* da base de dados, removendo as palavras cujas frequências são inferiores à percentagem mínima e superiores à máxima
Tfidf	-	Lê os valores da tabela *count* e calcula o seu *tf-idf*, guardando esses valores numa tabela de nome *tfidf*

Existe ainda o *script* que lida com a criação do ficheiro ARFF, referido anteriormente, que recebe os parâmetros descritos na Tabela 3.3:

Tabela 3.3: *Script* que cria o ficheiro ARFF

Name	*Input*	*Output*
Create ARFF	Ficheiro de texto com os pmid que queremos inserir no ficheiro ARFF	Ficheiro ARFF com os textos escolhidos

Por fim, foram criados alguns *scripts* que leem o ficheiro de *settings* e correm todos os algoritmos pretendidos, calculam o valor de cada palavra e criam o ficheiro ARFF como se mostra na Tabela 3.4:

Tabela 3.4: *Scripts* que correm todos os módulos do sistema

Name	*Input*	*Output*
Get Text	*Pmid* do texto que queremos retirar da base de dados Secção do texto que queremos retirar	Ficheiro de texto com a secção do texto de *pmid* escolhido
All Algorithms	-	Corre os algoritmos de TM escolhidos, pela ordem indicada ficheiro de *settings*
Select Texts	-	Seleciona o *X* textos (quantidade indicada no ficheiro de *settings*), aleatoriamente, retira-os da base de dados utilizando o *Get Text*, corre os algoritmos pretendidos utilizando o *All Algorithms*, conta a frequência das palavras dos textos retirados, corre o filtro, calcula o *tf-idf* e cria o ficheiro ARFF com os textos selecionados

Com estes módulos, o sistema pode ser facilmente utilizado e obter resultados finais de uma forma simples.

3.3 Sumário

Neste capítulo falamos do sistema desenvolvido nesta dissertação para testar diferentes abordágens de TM. O objetivo principal deste sistema é conseguir testar diferentes métodos de pré-processamento e avaliar os resultados obtidos. Neste sentido, a existência de vários módulos traz flexibilidade, mas deteriora a rapidez de execução do código, tornando estes testes longos e demorados. Este sistema foi utilizado para estudar diferentes formas de pré-processar os textos e avaliar os seus resultados. No próximo capítulo apresentam-se os resultados obtidos.

Capítulo 4

Um Caso de Estudo

Este capítulo descreve as experiências realizadas usando como caso de estudo textos curados do *corpus* TREC *genomics*. São descritos os diferentes métodos de pré-processamento e analisados os resultados dos algoritmos de classificação.

4.1 Conjunto de Dados

O conjunto de dados (TREC *genomics*) utilizado no caso de estudo foi originalmente disponibilizado na conferência TREC de 2005. Os dados foram guardados numa base de dados MySQL chamada *TrecCorpus*, contendo 11 880 documentos. O *script* que lida com a base de dados passa a secção do documento que pretendemos para um ficheiro de texto, como mostra a Figura 4.1.

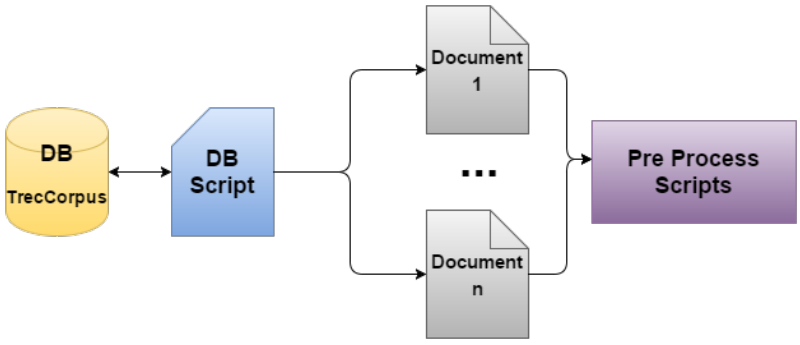

Figura 4.1: *Script* que lida com a base de dados

Cada documento possui um número de identificação (Pub Med InDex – *pmid*). O conteúdo dos documentos está separado por secções: *title*, *abstract*, *introduction*, *methods*, *results* e *conclusions*. Cada documento está classificado como relevante ou não relevante para duas categorias: *Allele* e *GO*. No caso de estudo foi utilizada a categoria *GO*. Os documentos estão classificados como **relevantes** ou **não relevantes** numa distribuição indicada na Tabela 4.1.

Tabela 4.1: Número de documentos relevantes e não relevantes no *dataset*

	Relevant	***Non Relevant***
GO	980	10900
Allele	670	11210

Para as experiências de classificação (o mesmo acontecendo nos dados originais), os dados estão divididos num conjunto de treino e num conjunto de teste com a distribuição indicada na Tabela 4.2.

Tabela 4.2: Distribuição das classes pelos conjuntos de treino e teste no *corpus* de 11 880 documentos.

Treino		**Teste**	
Relevant	***Non Relevant***	***Relevant***	***Non Relevant***
462	5375	518	5525

Como se pode observar na Tabela 4.2, o conjunto de dados é desequilibrado. Para o balancear, a classe maioritária foi amostrada num número de documentos igual à classe minoritária.

4.2 Pré-processamento Realizado

No pré-processamento dos documentos foram utilizados os seguintes algoritmos:

1. *Character Removal*

   Este módulo recebe um ficheiro com os caracteres que devem ser removidos do ficheiro de texto. O *output* é um ficheiro de texto sem os caracteres indesejados.

2. *Name Entity Recognition*

   O *script* que lida com os NER utiliza o ABNER para identificar entidades no texto. ABNER é uma ferramenta de *software* que identifica entidades da área de biologia, tais como: proteinas, DNA, RNA, *cell lines* e *cell types* [abn]. ABNER é uma ferramenta específica para a área da biologia e foi escolhida pois os textos fornecidos são dessa área.

   O módulo encontra as entidades e acrescenta *underscores* (_) ao NER, por exemplo, *lysosomal proteins* passa a *_lysosomal_proteins_*. Assim, um conjunto de palavras que referem uma entidade passam a ser contabilizadas como uma única palavra.

   O ABNER está implementado em *Java* e a sua biblioteca pode ser chamada de código *Java*. Como todo o nosso sistema está desenvolvido em *python*, o código *python* que reconhece entidades chama uma rotina de *Java* que identifica entidades no texto fornecido.

3. *Stop-Word Removal*

   Um módulo semelhante ao anterior, recebe um ficheiro com as palavras indesejadas e retira-as do texto.

4. Validação de Palavras

   Este módulo analisa todas as palavras do texto e verifica se eles pertencem ao dicionário do WordNET fornecido pelo NLTK. Todas as palavras que não existam são removidas.

5. Sinónimos

   Este módulo tem como objetivo encontrar palavras que sejam sinónimos e altera-as para a mesma palavra. Por exemplo, se tivermos estas três palavras no texto: "*like*", "*adore*" e "*love*", todas elas passam para "*like*", pois é o primeiro sinónimo destas palavras que apareceu. Para detetar os sinónimos é utilizada a biblioteca WordNET do NLTK. O WordNET contém listas de sínónimos para várias palavras. O *script* procura, para cada palavra, se já foi utilizado um sinónimos dessa palavra no texto e se sim, a palavra é subtituida pelo sinónimo utilizado, se não, a palavra mantém-se no texto e é guardada como sinónimo utilizado. Desta forma pode-se contabilizar melhor os sinónimos visto que passam a ser a mesma palavra, o que lhes dá um peso maior na contagem de palavras.

6. *Stemming*

   O módulo de *stemming* utiliza a biblioteca *stem* do NLTK para alterar as palavras para a sua raíz. O NLTK possui uma ferramenta chamada *stem* que contém uma lista de palavras e raízes de palavras. Assim, o processo de *stemming* é mais fácil e rápido.

Os algoritmos foram executados pela ordem referida. O primeiro algoritmo a correr é o *Character Removal* pois os caracteres não influenciam a informação que está nos documentos. O NER é o passo seguinte pois devem ser encontradas todas as entidades antes que o conteúdo dos documentos seja alterado. Depois disso, pode-se remover *stop-words*, validar as palavras e alterar as palavras para os seus sinónimos. O último algoritmo a ser corrido é o *stemming* pois se as palavras já estivessem "*stemmizadas*" os algritmos anteriores não executariam corretamente. Este fluxo está definido na Figura 4.2.

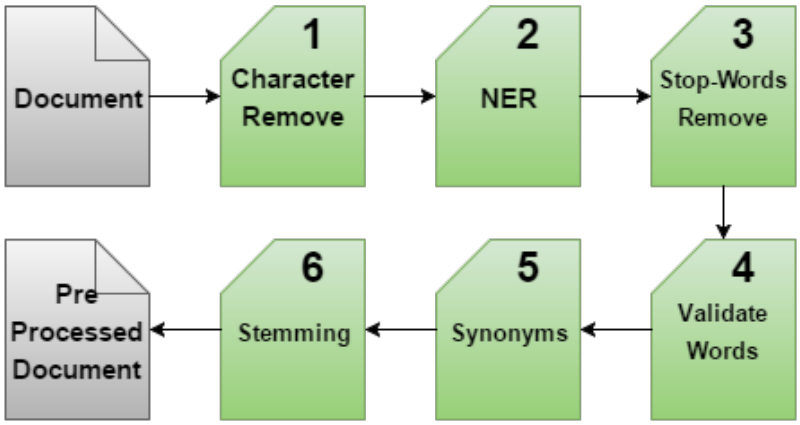

Figura 4.2: Fluxo de pré-processamento dos documentos

4.3 Classificação dos textos

O objectivo do estudo é encontrar um subconjunto das secções de um documento que permita obter resultados no processo de classificação de documentos, próximos, ou melhores que os resultados utilizando os documentos completos.

Tabela 4.3: conjuntos de subsecções utilizadas no estudo.

Identificador	**Conjunto de subsecções**
DS1	Título + Resumo
DS2	Título + Resumo + Métodos + Resultados
DS3	Título + Resumo + Métodos + Conclusões
DS4	Título + Resumo + Métodos + Resultados + Conclusões

Nas experiências realizadas foi utilizada a divisão original entre conjunto de treino e de teste. Os documentos originais foram divididos em secções e estas pré-processadas. Os subconjuntos de secções utilizados estão listados na Tabela 4.3. No final, o conjunto de dados foi representado no formato ARFF da ferramenta WEKA. Foi utilizada a versão *weka-3-7-13*.

Os quatro subconjuntos de secções da Tabela 4.3 originam os quatro conjuntos de dados da experiência. No conjunto de treino o número de exemplos classificados como *não relevantes* foram reduzidos por amostragem, ficando em número igual aos exemplos classificados como *relevantes*.

Foram utilizados os algoritmos Random Forest, LibSVM, J48, Naive Bayes e IBk do WEKA. Todos eles executaram com os valores dos parâmetros por omissão. A métrica utilizada para avaliar o desempenho foi a *Accuracy* no conjunto de treino.

4.4 Resultados

Os resultados obtidos são mostrados nas seguintes tabelas.

Tabela 4.4: Resultados obtidos pelos diferentes algoritmos no conjunto *DS1*.

Algoritmo	***Accuracy (%)***
RF	68.3
LibSVM	90.4
Naive Bayes	58.0
J48	70.0
IBk	79.1

A Tabela 4.4 mostra os resultados que os diferentes algoritmos obtiveram no conjunto *DS1*.

Tabela 4.5: Resultados obtidos pelos diferentes algoritmos no conjunto *DS2*.

Algoritmo	***Accuracy***
RF	72.3
LibSVM	90.3
Naive Bayes	58.7
J48	64.0
IBk	76.9

A Tabela 4.5 mostra os resultados que os diferentes algoritmos obtiveram no conjunto *DS2*.

Tabela 4.6: Resultados obtidos pelos diferentes algoritmos no conjunto *DS3*.

Algoritmo	***Accuracy***
RF	73.6
LibSVM	89.9
Naive Bayes	60.9
J48	65.7
IBk	75.5

A Tabela 4.6 mostra os resultados que os diferentes algoritmos obtiveram no conjunto *DS3*.

Tabela 4.7: Resultados obtidos pelos diferentes algoritmos no conjunto *DS4*.

Algoritmo	***Accuracy***
RF	73.2
LibSVM	90.0
Naive Bayes	62.8
J48	68.9
IBk	77.9

A Tabela 4.7 mostra os resultados que os diferentes algoritmos obtiveram no conjunto *DS4*.

4.5 Sumário

Os resultados mostram que a utilização do título e resumo permitem obter valores de *Accuracy* comparáveis com a utilização do documento completo. A confirmarem-se estes resultados a classificação de textos completos pode não trazer uma sobrecarga computacional adicional pois com um volume mais reduzido de informação conseguimos resultados semelhantes. Os resultados precisam, no entanto, de serem corroborados através da realização de mais experiências com diferentes conjuntos de documentos e algoritmos.

Capítulo 5

Conclusões

Nesta dissertação foi feita uma pesquisa sobre o estado da arte de metodologias de *Data Mining* e *Text Mining*. A partir da informação recolhida, foi criada uma ferramenta que executa algoritmos de *Text Mining* de modo a testar a viabilidade de utilizar texto completo no processo de classificação de documentos. Utilizando a ferramenta para pré-processar os documentos e criar os ficheiros ARFF e o WEKA para treinar os classificadores. Com o caso de estudo realizado pretendemos verificar se a utilização de menos secções de um documento no treino de um classificador podia obter resultados tão ou mais eficazes em comparação à utilização do texto completo.

Durante o pré-processamento dos documentos percebemos como o *Text Mining* é uma tecnologia que necessita de muito tempo de execução, acabando isso por atrasar a obtenção de resultados e, por consequência, a realização desta dissertação. Os resultados obtidos mostraram que a aplicação de algoritmos de *Text Mining* a textos completos tem resultados muito próximos à aplicação desses mesmos algoritmos apenas ao título e resumo (*abstract*) de documentos.

Em suma, os algoritmos de *Text Mining* são muito pesados computacionalmente e requerem muito tempo de execução. É possível reduzir um pouco o tempo de execução destes algoritmos de forem aplicados a uma quantidade menor de texto, em vez de os aplicar a textos completos. O nosso caso de estudo concluiu que os resultados obtidos através da utilização do texto completo são semelhantes aos resultados obtidos da aplicação dos mesmos algoritmos a apenas o título e resumo dos documentos. Além disso, os métodos tradicionais de *Text Mining* processam o texto como um saco de palavras (*bag of words*) ignorando a estrutura do texto, perdendo assim qualidade nos resultados que obtém. É assim necessário encontrar métodos que explorarem a estrutura da informação de um documento de forma a obter melhores resultados em algoritmos de *Text Mining*.

Referências

[abn] Abner official website. http://pages.cs.wisc.edu/~bsettles/abner/. Accessed: 2016-06-23.

[ARF] Weka wiki about arff. http://www.nltk.org/. Accessed: 2016-06-25.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. URL: http://dx.doi.org/10.1023/A:1010933404324, doi:10.1023/A:1010933404324.

[CG-15] *Multi-Relational Learning for Full Text Documents Classication*, 2015.

[dPVOG13] Célia Talma Martins de Pinho Valente Oliveira Gonçalves. *A Tool for Text Mining in Molecular Biology Domains*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, 2013.

[DUAdA06] Ramón Díaz-Uriarte e Sara Alvarez de Andrés. Gene selection and classification of microarray data using random forest. *BMC Bioinformatics*, 7(1):1–13, 2006. URL: http://dx.doi.org/10.1186/1471-2105-7-3, doi:10.1186/1471-2105-7-3.

[FH15] Ingo Feinerer e Kurt Hornik. *tm: Text Mining Package*, 2015. R package version 0.6-2. URL: http://CRAN.R-project.org/package=tm.

[FHM08] Ingo Feinerer, Kurt Hornik e David Meyer. Text mining infrastructure in r. *Journal of Statistical Software*, 25(5):1–54, March 2008. URL: http://www.jstatsoft.org/v25/i05/.

[Gat] Gate official website. https://gate.ac.uk/. Accessed: 2016-02-08.

[Hoo11] Apirak Hoonlor. *Sequential patterns and temporal patterns for text mining*. PhD thesis, Citeseer, 2011.

[HV09] William Hersh e Ellen Voorhees. Trec genomics special issue overview. *Inf. Retr.*, 12(1):1–15, February 2009. URL: http://dx.doi.org/10.1007/s10791-008-9076-6, doi:10.1007/s10791-008-9076-6.

[JaJ10] Joseph F. JaJa. Data mining practical machine learning tools and techniques, 2010. URL: http://www.umiacs.umd.edu/~joseph/classes/459M/year2010/Chapter5-testing-4on1.

[Joa98] Thorsten Joachims. Text categorization with suport vector machines: Learning with many relevant features. In *Proceedings of the 10th European Conference on Machine Learning*, ECML '98, pages 137–142, London, UK, UK, 1998. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=645326.649721.

REFERÊNCIAS

[luc] Apache lucene official website. https://lucene.apache.org/core/. Accessed: 2016-02-08.

[MN98] Andrew McCallum e Kamal Nigam. A comparison of event models for naive bayes text classification. In *IN AAAI-98 WORKSHOP ON LEARNING FOR TEXT CATEGORIZATION*, pages 41–48. AAAI Press, 1998.

[NLT] Nltk official website. https://weka.wikispaces.com/ARFF+%28book+version%29. Accessed: 2016-06-23.

[Por80] Martin F Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[Pow98] David M. W. Powers. Applications and explanations of zipf's law. In *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning*, NeMLaP3/CoNLL '98, pages 151–160, Stroudsburg, PA, USA, 1998. Association for Computational Linguistics. URL: http://dl.acm.org/citation.cfm?id=1603899.1603924.

[Pyt] Python official website. https://www.python.org/doc/. Accessed: 2016-06-23.

[Qui86] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. URL: http://dx.doi.org/10.1023/A:1022643204877, doi:10.1023/A:1022643204877.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[R] R official website. https://www.r-project.org/about.html. Accessed: 2016-02-08.

[RM] Rapidminer official website. https://rapidminer.com/us/. Accessed: 2016-02-08.

[Seb02] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, March 2002. URL: http://doi.acm.org/10.1145/505282.505283, doi:10.1145/505282.505283.

[Sta] Stanford parser official website. http://nlp.stanford.edu/software/lex-parser.shtml. Accessed: 2016-02-08.

[Vap99] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1999.

[WEK] Weka official website. http://www.cs.waikato.ac.nz/ml/weka/. Accessed: 2016-02-08.